IMPLEMENTATION OF MODIFIED VEDIC MULTIPLIER USING ON QUATERNARY SIGNED DIGIT NUMBER SYSTEM

[1]B DEVIKA RANI, [2]DR S GOVINDARAJULU
[1]M.Tech Student, [2]Professor
Department Of ECE
Dr.K.V.Subba Reddy Institute of Technology, Dupadu, Kurnool

ABSTRACT

The multiplier design which is initially developed for area optimizations is an array multiplier in which the main disadvantage is the worst-case delay. To overcome this disadvantage, many methodologies came into existence in which Vedic methodology has acquired prominence because of fast computations. It has been found that Vedic multiplier adopting Urdhva Tiryagbhyam is one of the most effective multipliers with minimum delay for multiplication of all types of numbers, either large or small. Though the speed of this multiplier is high, area occupancy is more. Thus, Binary to Excess-one Converter (BEC) technique is employed in this multiplier to reduce area. To further improve the performance, a 16×16 Vedic multiplier employing BEC adders and modified logic gates is developed.

The design doesn't require a radix conversion module as the sum is directly generated in binary using the novel concept of an adjusting bit. The proposed multiplier design is compared with a Vedic multiplier based on multi voltage or multi value logic [MVL], Vedic Multiplier that incorporates a QSD adder with a conversion module for quaternary to binary conversion, Vedic multiplier that uses Carry Select Adder and a commonly used fast multiplication mechanism such as Booth multiplier. All these designs have been developed using Verilog HDL and synthesized by Synopsys Design Compiler.

1. INTRODUCTION

BINARY multipliers are a widely used building block element in the design of microprocessors and embedded systems, and therefore, they are an important target for implementation optimization. Current implementations of binary multiplication follow the steps of :

1) recoding of the multiplier in digits in a certain number system;

2) digit multiplication of each digit by the multiplicand, resulting in a certain number of partial products;

3) reduction of the partial product array to two operands using multioperand addition techniques;

4) carry-propagate addition of the two operands to obtain the final result. The recoding type is a key issue, since it determines the number of partial products.

The usual recoding process recodes a binary operand into a signed-digit operand with digits in a minimally redundant digit set. Specifically, for radix-r ($r = 2m$), the binary operand is composed of non-redundant radix-r digits (by just making groups of m bits), and these are recoded from the set $\{0, 1,...,r - 1\}$ to the set $\{-r/2,..., -1, 0, 1, . . ., r/2\}$ to reduce the complexity of digit multiplications. For n-bit operands, a total of n/m partial products are generated for two's complement representation, and $(n + 1)/m$ for unsigned representation. Radix-4 modified Booth is a widely used recoding method, that recodes a binary operand into radix-4 signed digits in the set $\{-2, -1, 0, 1, 2\}$. This is a popular recoding since the digit multiplication step to generate the partial products only requires simple shifts and complementation.

The resulting number of partial products is about n/2. Higher radix signed recoding is less popular because the generation of the partial products requires odd multiples of the multiplicand which can not be achieved by means of simple shifts, but require carry-propagate additions. For instance, for radix-4signed digit recoding [9] the digit set is $\{-8, -7,..., 0,..., 7, 8\}$, so that some odd multiples of the multiplicand have to be generated. Specifically, it is required to generate $\times 3$, $\times 5$, and $\times 7$ multiples ($\times 6$ is obtained by simple shift of $\times 3$). The generation of each of these odd multiplies requires a two term addition or subtraction, yielding a total of three carry-propagate additions.

2. LITERATURE SURVEY:

However, the advantage of the high radix is that the number of partial products is further reduced. For instance, for radix-4and n-bit operands, about n/4 partial products are generated. Although less popular than radix-4, there exist industrial instances of radix-8. and radix-4multiplier in microprocessors implementations. The choice of these radices is related to area/delay/power optimization of pipelined multipliers (or fused multiplier adder as in the case of a Intel Itanium microprocessor), for balancing delay between stages and/or reduce the number of pipelining flip-flops.

A further consideration is that carry-propagate adders are today highly energy-delay optimized, while partial product reductions trees suffer the increasingly serious problems related to a complex wiring and glitching due to unbalanced signal paths. It is recognized in the literature that a radix-8 recoding leads to lower power multipliers compared to radix-4 recoding at the cost of higher latency (as a combinational block, without considering pipelining). Moreover, although the radix-4multiplier requires the generation of more odd multiples and has a more complex wiring for the generation of partial products, a recent microprocessor design considered it to be the best choice for low power (under the specific constraints for this microprocessor).

In some optimizations for radix-4 two's complement multipliers were introduced. Although for n-bit operands, a total of n/2 partial products are generated, the resulting maximum height of the partial product array is n/2 + 1 elements to be added (in just one of the columns). This extra height by a single-bit row is due to the +1 introduced in the bit array to make the two's complement of the most significant partial product (when the recoded most significant digit of the multiplier is negative). The maximum column height may determine the delay and complexity of the reduction tree, authors showed that this extra column of one bit could be assimilated (with just a simplified three bit addition) with the most significant part of the first partial product without increasing the critical path of the recoding and partial product generation stage. The result is that the partial product array has a maximum height of n/2. This reduction of one bit in the maximum height might be of interest for high-performance short-bit width two's complement multipliers (small n) with tight cycle time constraints, that are very common in SIMD digital signal processing applications. Moreover, if n is a power of two, the optimization allows to use only 4-2 carry-save adders for the reduction tree, potentially leading to regular layouts. These kind of optimizations can become particularly important as they may add flexibility to the "optimal" design of the pipelined multiplier.

Optimal pipelining in fact, is a key issue in current and future multiplier (or multiplier-add) units: 1) the latency of the pipelined unit is very important, even for throughput oriented applications, as it impacts the energy consumption of the whole core; and 2) the placement of the pipelining flip-flops should at the same time minimize total power, due to the number of flip-flops required and the unbalanced signal propagation paths. The methods proposed in were mostly focused on two's complement radix-4 Booth multipliers, thus leaving open the research and extension to higher radices and unsigned multiplications (for unsigned integer arithmetic or mantissa times mantissa in a floating-point unit). For a radix higher than 4, it is necessary to generate the odd multiples (usually with adders), resulting in the reduction of the time slacks necessary to "hide" the simplified three bit assimilation. Unsigned multiplication may produce a positive carry out during recoding (this depends of the value of n and the radix used for recoding), leading to one additional row, increasing the maximum height of the partial product array by one row, not just in one but in several columns. For all these reasons, we need to extend the techniques presented.

The challenge of the verifying a large design is growing exponentially. There is a need to define new methods that makes functional verification easy. Several strategies in the recent years have been proposed to achieve good functional verification with less effort. Recent advancement towards this goal is methodologies. The methodology defines a skeleton over which one can add flesh and skin to their requirements to achieve functional verification.

Complex multiplication is of immense importance in Digital Signal Processing (DSP) and Image Processing (IP). To implement the hardware module of Discrete Fourier Transformation (DFT), Discrete Cosine Transformation (DCT), Discrete Sine Transformation (DST) and modem broadband communications; large numbers of complex multipliers are required. Complex number multiplication is performed using four real

number multiplications and two additions/ subtractions. In real number processing, carry needs to be propagated from the least significant bit (LSB) to the most significant bit (MSB) when binary partial products are added [1]. Therefore, the addition and subtraction after binary multiplications limit the overall speed. Many alternative method had so far been proposed for complex number multiplication [2-7] like algebraic transformation based implementation[2], bit-serial multiplication using offset binary and distributed arithmetic [3], the CORDIC (coordinate rotation digital computer) algorithm [4], the quadratic residue number system (QRNS) [5], and recently, the redundant complex number system (RCNS). Blahut et. al [2] proposed a technique for complex number multiplication, where the algebraic transformation was used. This algebraic transformation saves one real multiplication, at the expense of three additions as compared to the direct method implementation. A left to right array [7] for the fast multiplication has been reported in 2005, and the method is not further extended for complex multiplication. But, all the above techniques require either large overhead for pre/postprocessing or long latency. Further many design issues like as speed, accuracy, design overhead, power consumption etc., should not be addressed for fast multiplication [8]. In algorithmic and structural levels, a lot of multiplication techniques had been developed to enhance the efficiency of the multiplier; which encounters the reduction of the partial products and/or the methods for their partial products addition, but the principle behind multiplication was same in all cases. Vedic Mathematics is the ancient system of Indian mathematics which has a unique technique of calculations based on 16 Sutras (Formulae). "Urdhva-tiryakbyham" is a Sanskrit word means vertically and crosswise formula is used for smaller number multiplication. "Nikhilam Navatascaramam Dasatah" also a Sanskrit term indicating "all from 9 and last from 10", formula is used for large number multiplication and subtraction. All these formulas are adopted from ancient Indian Vedic Mathematics. In this work we formulate this mathematics for designing the complex multiplier architecture in transistor level with two clear goals in mind such as: i) Simplicity and modularity multiplications for VLSI implementations and ii) The elimination of carry propagation for rapid additions and subtractions. Mehta et al. [9] have been proposed a multiplier design using "Urdhva-tiryakbyham" sutras, which was adopted from the Vedas. The formulation using this sutra is similar to the modem array multiplication, which also indicating the carry propagation issues. A multiplier design using "Nikhilam Navatascaramam Dasatah" sutras has been reported by Tiwari et. al [10] in 2009, but he has not implemented the hardware module for multiplication.

Multiplier implementation in the gate level (FPGA) using Vedic Mathematics has already been reported but to the best of our knowledge till date there is no report on transistor level (ASIC) implementation of such complex multiplier. By employing the Vedic mathematics, an N bit complex number multiplication was transformed into four multiplications for real and imaginary terms of the final product. "Nikhilam. Navatascaramam Dasatah" sutra is used for the multiplication purpose, with less number of partial products generation, in comparison with array based multiplication. When compared with existing methods such as the direct method or the strength reduction technique, our approach resulted not only in simplified arithmetic operations, but also in a regular arraylike structure. The multiplier is fully parameterized, so any configuration of input and output word-lengths could be elaborated. Transistor level implementation for perfonnance parameters such as propagation delay, dynamic leakage power and dynamic switching power consumption calculation of the proposed method was calculated by spice spectre using 90 nm standard CMOS technology and compared with the other design like distributed arithmetic[3], parallel adder based implementation [1] and algebraic transfonnation[2] based implementation. The calculated results revealed (16,16)x(16,16) complex multiplier have propagation delay only 4 ns with 6.5 mW dynamic switching power.
In this paper we report on a novel high speed complex multiplier design using ancient Indian Vedic mathematics.

3. VEDIC MULTIPLICATION ALGORITHMS

Vedic mathematics is part of four Vedas (books of wisdom). It is part of Sthapatya-Veda (book on civil engineering and architecture), which is an upa-veda (supplement) of Atharva Veda. It covers

explanation of several modern mathematical terms including arithmetic, geometry (plane, co-ordinate), trigonometry, quadratic equations, factorization and even calculus. His Holiness Jagadguru Shankaracharya Bharati Krishna Teerthaji Maharaja (1884-1960) comprised all this work together and gave its mathematical explanation while discussing it for various applications. Swahiji constructed 16 sutras (formulae) and 16 Upa sutras (sub formulae) after extensive research in Atharva Veda. Obviously these formulae are not to be found in present text of Atharva Veda because these formulae were constructed by Swamiji himself. Vedic mathematics is not only a mathematical wonder but also it is logical. That‟s why VM has such a degree of eminence which cannot be disapproved. Due these phenomenal characteristic, VM has already crossed the boundaries of India and has become a leading topic of research abroad. VM deals with several basic as well as complex mathematical operations. Especially, methods of basic arithmetic are extremely simple and powerful.

The word „Vedic‟ is derived from the word „veda‟ which means the store-house of all knowledge. Vedic mathematics is mainly based on 16 Sutras (or aphorisms) dealing with various branches of mathematics like arithmetic, algebra, geometry etc. These Sutras along with their brief meanings are enlisted below alphabetically.

1) (Anurupye) Shunyamanyat – If one is in ratio, the other is zero.
2) Chalana-Kalanabyham – Differences and Similarities.
3) Ekadhikina Purvena – By one more than the previous One.
4) Ekanyunena Purvena – By one less than the previous one.
5) Gunakasamuchyah – The factors of the sum is equal to the sum of the factors.
6) Gunitasamuchyah – The product of the sum is equal to the sum of the product.
7) Nikhilam Navatashcaramam Dashatah – All from 9 and last from 10.
8) Paraavartya Yojayet – Transpose and adjust.
9) Puranapuranabyham – By the completion or noncompletion.
10) Sankalana- vyavakalanabhyam – By addition and by subtraction.
11) Shesanyankena Charamena – The remainders by the last digit.
12) Shunyam Saamyasamuccaye – When the sum is the same that sum is zero.
13) Sopaantyadvayamantyam – The ultimate and twice the penultimate.
14) Urdhva-tiryakbhyam – Vertically and crosswise.
15) Vyashtisamanstih – Part and Whole.
16) Yaavadunam – Whatever the extent of its deficiency.

These methods and ideas can be directly applied to trigonometry, plain and spherical geometry, conics, calculus (both differential and integral), and applied mathematics of various kinds. As mentioned earlier, all these Sutras were reconstructed from ancient Vedic texts early in the last century. Many Sub-sutras were also discovered at the same time, which are not discussed here.

The beauty of Vedic mathematics lies in the fact that it reduces the otherwise cumbersome-looking calculations in conventional mathematics to a very simple one. This is so because the Vedic formulae are claimed to be based on the natural principles on which the human mind works. This is a very interesting field and presents some effective algorithms which can be applied to various branches of engineering such as computing and digital signal processing.

The multiplier architecture can be generally classified into three categories. First is the serial multiplier which emphasizes on hardware and minimum amount of chip area. Second is parallel multiplier (array and tree) which carries out high speed mathematical operations. But the drawback is the relatively larger chip area consumption. Third is serial-parallel multiplier which serves as a good trade-off between the times consuming serial multiplier and the area consuming parallel multipliers.

ALGORITHMS OF VEDIC MATHEMATICS:-

VEDIC MULTIPLICATION

The proposed Vedic multiplier is based on the Vedic multiplication formulae (Sutras). These Sutras have been traditionally used for the multiplication of two numbers in the decimal number system. In this work, we apply the same ideas to the binary number system to make the proposed algorithm compatible with the digital hardware. Vedic multiplication based on some algorithms, some are discussed below:

Urdhva Tiryakbhyam sutra

The multiplier is based on an algorithm Urdhva Tiryakbhyam (Vertical & Crosswise) of ancient Indian Vedic Mathematics. Urdhva Tiryakbhyam Sutra is a general multiplication formula applicable to all cases of multiplication. It literally means "Vertically and crosswise". It is based on a novel concept through which the generation of all partial products can be done with the concurrent addition of these partial products. The parallelism in generation of partial products and their summation is obtained using Urdhava Triyakbhyam explained in fig 2.1. The algorithm can be generalized for n x n bit number. Since the partial products and their sums are calculated in parallel, the multiplier is independent of the clock frequency of the processor. Thus the multiplier will require the same amount of time to calculate the product and hence is independent of the clock frequency. The net advantage is that it reduces the need of microprocessors to operate at increasingly high clock frequencies. While a higher clock frequency generally results in increased processing power, its disadvantage is that it also increases power dissipation which results in higher device operating temperatures. By adopting the Vedic multiplier, microprocessors designers can easily circumvent these problems to avoid catastrophic device failures. The processing power of multiplier can easily be increased by increasing the input and output data bus widths since it has a quite a regular structure. Due to its regular structure, it can be easily layout in a silicon chip. The Multiplier has the advantage that as the number of bits increases, gate delay and area increases very slowly as compared to other multipliers. Therefore it is time, space and power efficient. It is demonstrated that this architecture is quite efficient in terms of silicon area/speed.

1) Multiplication of two decimal numbers-325*738

To illustrate this multiplication scheme, let us consider the multiplication of two decimal numbers (325 * 738). Line diagram for the multiplication is shown in Fig.2.2. The digits on the both sides of the line are multiplied and added with the carry from the previous step. This generates one of the bits of the result and a carry. This carry is added in the next step and hence the process goes on. If more than one line are there in one step, all the results are added to the previous carry. In each step, least significant bit acts as the result bit and all other bits act as carry for the next step. Initially the carry is taken to be zero. To make the methodology more clear, an alternate illustration is given with the help of line diagrams in figure 2.2 where the dots represent bit „0" or „1".

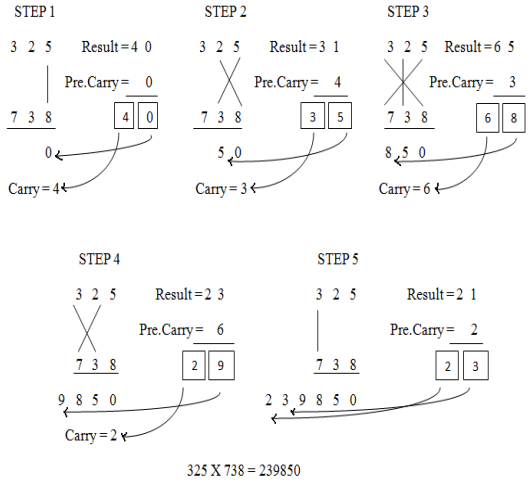


Figure.1: Multiplication of two decimal numbers by Urdhva Tiryakbhyam.

4. PROPOSED METHOD

Any proposed system must be efficient in terms of power, speed and size as per growing technology. In early days Vedic mathematics is based on 16 vedic sutras.By using Vedic methods the mathematical operations are fast and the processing speed to perform the operations can be improved. There has been many existing binary multipliers which are efficient.

MULTIPLIER

A binary multiplier [3] can be used in digital electronics as a electronic circuit,such as in computers to find the product of two binary numbers.Carbon-copy of normal multiplication technique is used by binary multiplier,the multiplicand is multiplied with each bit of the multiplier beginning from the least significant bit.Two half adder(HA) modules can be used in order to implement a 2-bit binary multiplier.A no of computer arithmetic calculations can be used to appliance digital multiplier.Among these techniques many imply computing a set of partial products, and then summing the generated partial products together.Fig. 1, shows 2x2 binary multiplier.

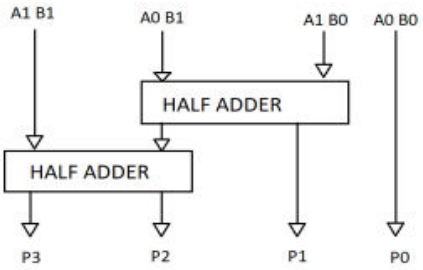


Fig. 2. 2x2 Binary Multiplier

A. Ripple Carry Adder(RCA)

In a multiplier number of Full adders are arranged in a manner to give the results of an addition operation of n-bit binary sequence.The input to next Full adder stage is obtained from the previous carry output of adder,it repeats until it reaches to the ending stage.Fig. 2 shows Four bit(RCA) Ripple Carry Adder [4].

VEDIC MULTIPLIER

The mode used by Vedic multiplier [6] is Vedic mathematics.By using this technique it will increase,and consumes fewer hardware elements.The sutra [6] used by Vedic multiplier is Urdhva Tiryakbhyam[3] which means Vertically as well as Crosswise. The Fig. 3 shows block diagram of 32 bit vedic multiplier circuit. The2 input bits are separated into 2 similar parts the vertical and cross product calculations can be done as shown in Fig. 3, with inputs A[31:0] and B[31:0].As shown in the Fig. 3, the 2 adders are used in the design of intermediate stages of the addition.The output carry Cout from these two adders is given as input to another RCA.If bits are not of equal sizes concatenate them.For 32-bit Modified Vedic multiplier the outputs of parallel adder is given to OR gate and of the size of last RCA is reduced to half. Fig. 3, shows 32-bit Vedic multiplier.

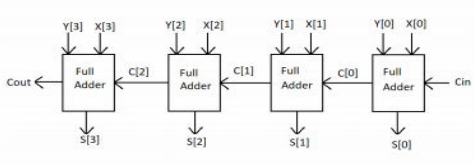


Fig. 3. 4-Bit Ripple Carry Adder

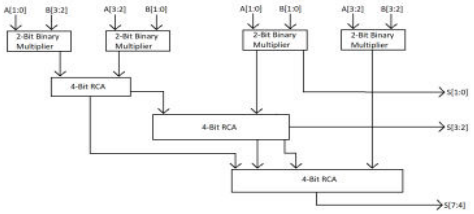
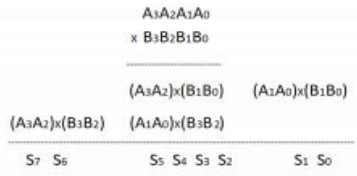


Fig. 4. 4-Bit Vedic Multiplier

MODIFIED VEDIC MULTIPLIER

In the proposed paper, the two parallel adders are replaced by CSA [4] for the better execution of the multiplier architecture. The recommended modified Vedic multiplication methodology is done in the following for 4 bit inputs, A(A3 -A0) and B(B3 -B0) and 8 bit output S (S7 -S0).



A multiplier of 2 bit is used to calculate intermediate stage results, and the output is 4 bits. (A3A2)(B3B2) using 2 bit multiplier generates result: S33S32S31S30 (A3A2)(B1B0) using 2 bit multiplier generates result: S23S22S21S20 (A1A0)(B3B2) using 2 bit multiplier generates result: S13S12S11S10 (A1A0)(B1B0) using 2 bit multiplier generates result: S03S02S01S00

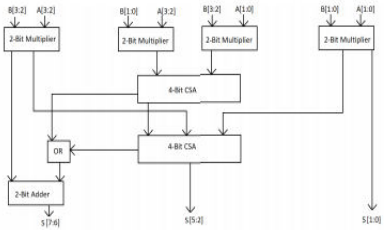


Fig. 5. Modified 4 Bit Vedic Multiplier

The 4 bit CSA Carry Save Adder [4] is used to add three 4 bit data inputs: S23S22S21S20, S13S12S11S10 and S31S30 S03 S02. The proposed 4 bit modified Vedic multiplier [7] is designed and the Fig. 4 shows it. The last two MSBs of CSA outputs are given as inputs to OR gate. In addition, the last stage 4 bit RCA is replaced by 2 bit adder circuit through which the output value of OR gate can be controlled. One of the input to last stage 2-bit adder[6] is obtaining from the output of or gate. Similarly, a 4 bit RCA block is a must needed for 8 bit vedic multiplication design. Fig. 5, shows 32 bit modified Vedic multiplier.

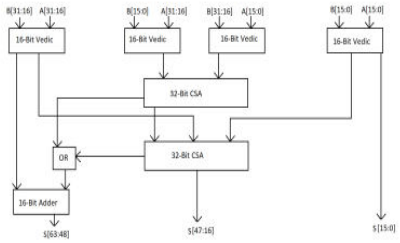


Fig. 6. Modified 32 Bit Vedic Multiplier

5. EXTENSION

One of the primary features that help us determine the computational power of a processor is the speed of its arithmetic unit. An important function of an arithmetic block is multiplication because, in most mathematical computations, it forms the bulk of the execution time. Thus, the development of a fast multiplier has been a key research area for a long time. Some of the important algorithms proposed for fast multiplication in literature are Array, Booth and Wallace multipliers [1]-[5]. Vedic Mathematics [6, 7] is a methodology of arithmetic rules that allows for more efficient implementations regarding speed. Multiplication in this methodology consists of three steps: generation of partial products, reduction of partial products, and finally carrypropagate addition. Multiplier design based on Vedic mathematics has many advantages as the partial products and sums are generated in one step, which reduces the carry propagation from LSB to MSB. This feature helps in scaling the design for larger inputs without proportionally increasing the propagation delay as all smaller blocks of the design work concurrently. References [8], [9] and [11] compared Vedic Multiplier with other multiplier architectures namely Booth, Array and Wallace on the basis of delay and power consumption. Vedic multiplier showed improvements in both the parameters over other architectures. Thus, many implementations of multiplication algorithms based on Vedic sutras have been reported in literature [10]-[12]. Vedic multiplier schemes proposed in literature are based on Urdhva Tiryagbhyam and Nikhilam sutras of Vedic Mathematics. As Nikhilam sutra is only efficient for inputs that are close to the power of 10, in this paper a design to perform high-speed multiplication based on the Urdhva Tiryagbhyam sutra of Vedic Mathematics which is generalized method for all numbers, has been presented. The final step, carry-propagate addition, requires a fast adder scheme because it forms a part of the critical path. A variety of adder schemes have been proposed in literature to optimize the performance of Vedic multiplier [13]. Adder based on QSD shows an improvement in speed over other state of the art adders [14, 15]. Earlier implementations of QSD adder were based on Multi Voltage or Multi Value Logic (MVL) [16]. The difficulty in application of quaternary addition outside MVL (Multi Voltage logic) is that, the adder is only a small unit of the design whose outputs will needed to be converted back to binary for further processing. However, use of a conversion module undermines the advantages gained in speed by using QSD. In this paper, a novel implementation of an adder based on QSD is proposed, which reduces the carry propagation delay in the design by making use of carry free arithmetic. The proposed adder design works on a hybrid of binary and quaternary number systems wherein the sum is directly generated in binary using the concept of an adjusting bit, eliminating the conversion module. The design can be scaled to larger bit implementations such as 32, 64, 128 or more with minimal increase in propagation delay owing to the parallelism prevalent in the design. We have compared our design with a Vedic multiplier based on MVL logic that uses a ripple carry adder [16], Vedic Multiplier that incorporates a QSD adder and a conversion module for quaternary to binary conversion, Vedic multiplier that uses state of the art fast adder scheme such as Carry select adder [17] and a commonly used fast multiplication mechanism such as Booth multiplier [18], to prove the feasibility of our design across important comparison points.

Quaternary Signed Digit (QSD number system)

The QSD is a radix-4 number system that provides the benefit of faster arithmetic calculations over binary computation, as it eliminates rippling of carry during addition. Every number in QSD can be represented using digits from the set {-3,-2,-1, 0, 1, 2, 3}. Being a higher radix number system it utilizes less number of gates and hence saves on time and reduces circuit complexity. The stages involved in addition of two numbers in QSD are: Stage1: Generation of intermediate carry and sum: When two digits are added in QSD

number system, the resulting sum ranges between -6 to +6. Numbers with magnitude higher than 3 are represented by multiple digits with least significant digit representing sum and the next digit corresponds to carry. Also, every number in QSD can have multiple representations [14, 15]. The representation is chosen such that the magnitude of sum digit is 2 or less than 2 and the magnitude of carry digit is 1 or less than 1, the reason for which is explained in the next stage. Stage2: The intermediate sum and carry have a limit fixed on their magnitude because this allows carry free addition in the second step. The result can be obtained directly by adding the sum digit with the carry of the lower significant digit [14, 15].

PROPOSED DESIGN

A. 4x4 Multiplier

Block diagram of a 4x4 multiplier is shown in Fig. 3. In this multiplier, four 2x2 multipliers are arranged systematically. Each multiplier accepts four input bits; two bits from multiplicand and other two bits from multiplier. Addition of partial products is done using two four bit Quaternary adders, a two-bit adder and a half adder. The final result is obtained by concatenating the least significant two bits of the first multiplier, four sum bits of the second four-bit Quaternary adder and the sum bits of two-bit adder.

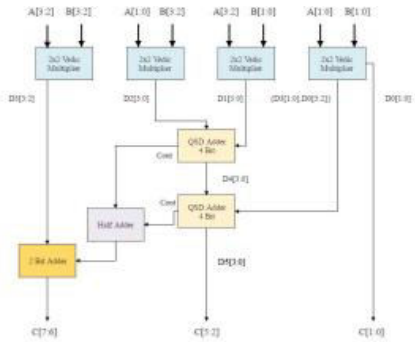


Fig. 3. Proposed 4x4 Multiplier

Table I shows all intermediate and final results involved in the multiplication process of two binary numbers, A = (1111)2 and B = (1001)2. The data flow in the proposed 4x4 multiplier is given below: 1) A[1:0] and B[1:0], A[3:2] and B[1:0], A[1:0] and B[3:2], and A[3:2] and B[3:2] are multiplied by 2x2 Vedic multipliers, giving output D0[3:0], D1[3:0], D2[3:0] and D3[3:0] respectively. 2) D1 [3:0] and D2[3:0] are added by the proposed 4 bit QSD adder, giving D4[3:0] and a carry out as the outputs. 3) D4[3:0] and {D3[1:0], D0[3:2]} are added by the second 4 bit QSD adder, giving D5[3:0] and a carry out as the outputs. 4) The half adder is used to add the carry outs of the QSD adders. The output obtained is fed to the 2 Bit Adder along with D3[3:2]. 5) The result, C, in binary is obtained by concatenation of output of 2 Bit Adder, D5[3:0] and D0[1:0]. The proposed design can be extended to multiply both negative and positive integers by an addition of a sign bit in both inputs. An XOR logic can then be used to compute the sign bit of the final output. The multiplication of the magnitudes will proceed simultaneously in a similar manner to the example described above.

TABLE I. MULTIPLICATION RESULT OF TWO 4 BIT BINARY NUMBERS USING THE PROPOSED DESIGN

	Binary equivalent	Decimal equivalent	Explanation
A	$(1111)_2$	15	Input 1
B	$(1001)_2$	9	Input 2
D0	$(0011)_2$	3	Output of 2x2 Vedic Multiplier 1
D1	$(0011)_2$	3	Output of 2x2 Vedic Multiplier 2
D2	$(0110)_2$	6	Output of 2x2 Vedic Multiplier 3
D3	$(0110)_2$	6	Output of 2x2 Vedic Multiplier 4
D4	$(01001)_2$	9	Output of 4 bit QSD adder 1 (D1+D2)
D5	$(10001)_2$	17	Output of 4 bit QSD adder 2 (D4 +{D3[1:0],D0[3:2]})
C[1:0]	$(11)_2$	3	D0[1:0]
C[5:2]	$(0001)_2$	1	D5[3:0]
C[7:6]	$(10)_2$	2	Output of 2 Bit Adder (D3[3:2]+D4[4]+D5[4])
C[7:0]	$(10000111)_2$	135	Final Result

B. 32x32 multiplier

The 4x4 multiplier design can be scaled to multiply larger numbers as shown in Fig. 4, where the design is scaled up for a 32 bit multiplier.

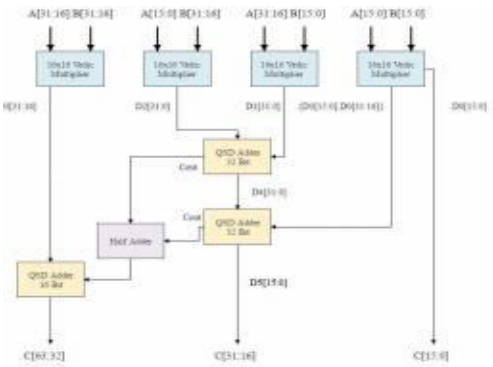


Fig. 4. Proposed 32x32 Multiplier

C. Proposed adder design based on QSD

In this paper, a novel idea of an adder, based on QSD (Quaternary Signed Digit) is proposed. The algorithm for the proposed adder uses a hybrid of quaternary and binary number systems. The outputs from smaller multipliers are obtained as binary strings. Inside the addition module, this string is broken into quaternary digits of two bits each. Addition using QSD allows us to reduce the carry propagation delay by making use of carry free arithmetic i.e. the carry doesn't ripple past the subsequent quaternary digit. Especially for higher bit input strings this method is extremely efficient. The difficulty in application of quaternary addition outside MVL (Multi Voltage logic) is that the least significant 2 bits of the binary representation of the quaternary digits can't be directly concatenated to form an output binary string for every case as depicted in Table II. Each string would have to be read individually and a conversion module that converts quaternary to binary would have to be employed. To overcome this limitation, the concept of an adjusting bit has been introduced.

TABLE II. CONVERSION OF A QUATERNARY NUMBER TO BINARY NUMBER SYSTEM

Quaternary number A	2 1 → 010_001	Quaternary number B	2 $\bar{1}$ → 010_111
Binary equivalent of A	1001	Incorrect Binary equivalent of B	1011
Decimal equivalent of A	9	Incorrect Decimal equivalent of B	11

The Intermediate sum lies in the range [0, 6], as the operands are unsigned numbers. From [16], for quaternary addition to be carry free beyond the first stage, the intermediate sum can't be greater than 2. To ensure this stipulation holds true, the (1)4 representation of 3 needs to be chosen while adding. However, this represents a blocking case when converting the final output string back into binary as it prohibits us from simply concatenating the lower two bits of quaternary output strings to get the binary equivalent. For addition of unsigned numbers, if the (03)4 representation would have been used, direct concatenation of results could have been possible. But, then that wouldn't have always been carry free after the initial stage. Thus, the concept of an adjusting bit has been devised to solve the dilemma of which representation of 3 to use, such that both carry free addition and concatenation of output string bits to get the final output can be realized in the same design. The solution to the problem described above, is that the (03)4 representation of 3 is required to be taken instead of the (1)4 representation in some cases. But, determining when such a change is required before proceeding with the addition will increase the delay of the design and be counter-productive. Thus, the (1)4 representation of 3 is always selected in stage 1, to satisfy necessary conditions for carry free arithmetic. While necessary adjustments are made in stage 2 if (03)4 representation was to be taken, the need for such an adjustment is determined via an adjusting bit.

TABLE V. EXAMPLE OF QUATERNARY ADDITION USING PROPOSED LOGIC

A	$(1100)_2$	Input 1
B	$(0011)_2$	Input 2
Q1	11_00 → 3 0	Quaternary representation of Binary number A
Q2	00_11 → 0 3	Quaternary representation of Binary number B
X_2X_1	3 0 → 011_000	Input A
Y_2Y_1	0 3 → 000_011	Input B
U1	$\bar{1}\bar{1}$ → 111_111	Stage 1 output (Intermediate sum)
U2	1 1	Stage 1 output (Intermediate carry)
S2	1	2nd intermediate sum digit is 3
S1	1	1st intermediate sum digit is 3
S0	0	0th digits do not exist
C1	1	Carry from sum of 2nd digits is 1
C0	1	Carry from sum of 1st digits is 1
A2	S2.(S1+$\overline{C1}$) = 1	2nd Adjusting Bit
A1	S1.(S0 + $\overline{C0}$) = 0	1st Adjusting Bit
A	1 0	Stage 1 output (Adjusting Bit)
U3	001_111_111	Stage 2 output (Before Adjusting Bit logic)
U4	000_111_111	Stage 2 output (After Adjusting Bit logic)
R	$(1111)_2$	Result after concatenation

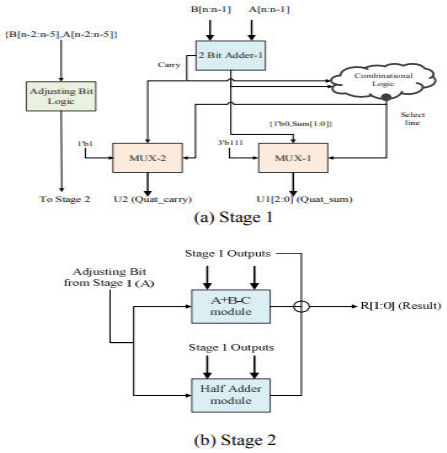


Fig. 5. Proposed Adder

Where Sn-2 is true if n-2th intermediate sum digit is 3. This formula can cover the problem

of n consecutive 3's in a similar manner. The adjusting bit can be predicted based on the initial inputs to the adders itself. It can be computed in parallel with Stage 1. Thus, effect on delay of the adder is minimal. The above example is revaluated with the modified formula: Input A= (X3X2X1)4 = (A8A7A6A5A4A3A2A1A0)2 = (030)4 Input B = (Y3Y2Y1)4 = (B8B7B6B5B4B3B2B1B0)2 = (003)4 Adjusting Bit for addition of Xn and Yn is Sn-1.(Sn-2+). As can be seen from the flow of data shown in Table V. The modified formula gives the correct binary output after concatenation. The proposed adder works in two stages, as shown in Fig. 5. 1) In the first stage, as in Fig. 5(a), every individual digit at the same position in the quaternary representation of two n-bit numbers A and B is added using a 2 Bit Adder to generate a sum. This sum lies in the range [0, 6]. From the sum obtained from the adder, the intermediate sum and intermediate carry for the next stage are calculated in parallel using 2x1 multiplexers. The logic for the selection of the representation of sum and carry has been explained in [16]. The adjusting bit is also computed in parallel with the addition process. The input to the adjusting bit calculation block for every quaternary digit addition are the previous two quaternary digits of A and B signified by [n-2: n-5]. 2) Second stage has two modules as shown in Fig. 5(b). One is a one-bit module that performs the computation (A+BC). In this case A would be LSB of intermediate sum, B would be carry from the previous quaternary digit addition and C would be the adjusting bit. The other module will be a half adder which will add the carry from the (A+B-C) module and the bit to the left of the least significant bit of the intermediate sum. As for the final concatenation, the sign bit would not be used owing to the adjustments proposed in the design. Thus, its final value is not computed.

6. SIMULATION RESULTS

Proposed results

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice LUTs	1729	17600	9%
Number of fully used LUT-FF pairs	0	1729	0%
Number of bonded IOBs	128	100	128%

6.1.Design summary

```
MUXCY:CI->O     1  0.013  0.000  CPA2/Madd_fsum_cy<56> (CPA2/Madd_fsum_cy<56>)
MUXCY:CI->O     1  0.013  0.000  CPA2/Madd_fsum_cy<57> (CPA2/Madd_fsum_cy<57>)
MUXCY:CI->O     1  0.013  0.000  CPA2/Madd_fsum_cy<58> (CPA2/Madd_fsum_cy<58>)
MUXCY:CI->O     1  0.013  0.000  CPA2/Madd_fsum_cy<59> (CPA2/Madd_fsum_cy<59>)
MUXCY:CI->O     1  0.013  0.000  CPA2/Madd_fsum_cy<60> (CPA2/Madd_fsum_cy<60>)
XORCY:CI->O     3  0.251  0.438  CPA2/Madd_fsum_xor<61> (n0538<61>)
LUT3:I0->O      2  0.043  0.347  CPA5/aa[61].b1/CCC1/Mmux_cout111 (CPA5/C<62>)
LUT5:I3->O      1  0.043  0.550  CPA5/aa[62].b1/CCC2/Mmux_sum11 (CPA5/sum<62>)
LUT6:I0->O      1  0.043  0.000  CPA5/Madd_fsum_lut<62> (CPA5/Madd_fsum_lut<62>)
MUXCY:S->O      0  0.230  0.000  CPA5/Madd_fsum_cy<62> (CPA5/Madd_fsum_cy<62>)
XORCY:CI->O     1  0.251  0.279  CPA5/Madd_fsum_xor<63> (p_63_OBUF)
OBUF:I->O            0.000        p_63_OBUF (p<63>)
----------------------------------------
Total                  6.410ns (2.362ns logic, 4.048ns route)
                              (36.8% logic, 63.2% route)
```

Time summary

6.2. Power summary

Extension Results

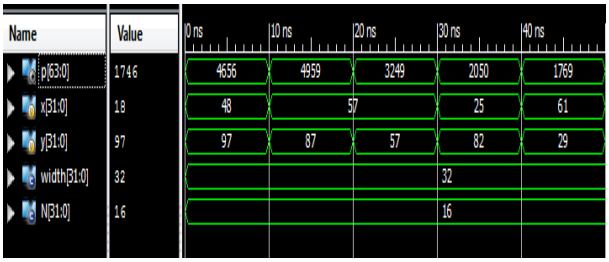


Figure 6.3. Simulation outcome.

Figure 6.3 shows the simulation results of proposed DST-R4BM. Here, width is the input pin, which is used to change the size of DST-R4BM. So, variable width concept is justified. Further, X and Y are the input data ports and P is the output port.

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice LUTs	1729	17600	9%
Number of fully used LUT-FF pairs	0	1729	0%
Number of bonded IOBs	128	100	128%

Figure.6 4. Design summary.

Figure 6.4 shows the design (area) summary of proposed method. Here, the proposed method utilizes the low area in terms of slice LUTs i.e., 1729 out of available 17600.

```
    MUXCY:CI->O           1  0.013   0.000  CPA2/Madd_fsum_cy<56> (CPA2/Madd_fsum_cy<56>)
    MUXCY:CI->O           1  0.013   0.000  CPA2/Madd_fsum_cy<57> (CPA2/Madd_fsum_cy<57>)
    MUXCY:CI->O           1  0.013   0.000  CPA2/Madd_fsum_cy<58> (CPA2/Madd_fsum_cy<58>)
    MUXCY:CI->O           1  0.013   0.000  CPA2/Madd_fsum_cy<59> (CPA2/Madd_fsum_cy<59>)
    MUXCY:CI->O           1  0.013   0.000  CPA2/Madd_fsum_cy<60> (CPA2/Madd_fsum_cy<60>)
    XORCY:CI->O           3  0.251   0.438  CPA2/Madd_fsum_xor<61> (n0538<61>)
    LUT3:I0->O            2  0.043   0.347  CPA5/aa[61].b1/CCC1/Mmux_cout111 (CPA5/C<62>)
    LUT5:I3->O            1  0.043   0.550  CPA5/aa[62].b1/CCC2/Mmux_sum11 (CPA5/sum<62>)
    LUT6:I0->O            1  0.043   0.000  CPA5/Madd_fsum_lut<62> (CPA5/Madd_fsum_lut<62>)
    MUXCY:S->O            0  0.230   0.000  CPA5/Madd_fsum_cy<62> (CPA5/Madd_fsum_cy<62>)
    XORCY:CI->O           1  0.251   0.279  CPA5/Madd_fsum_xor<63> (p_63_OBUF)
    OBUF:I->O                0.000          p_63_OBUF (p<63>)
    ----------------------------------------
    Total                    6.410ns (2.362ns logic, 4.048ns route)
                                     (36.8% logic, 63.2% route)

=========================================================================
```

Figure 6.5. Time summary

Figure 6.5 shows the time summary of proposed method. Here, the proposed method consumed total 6.410ns of time delay, where 2.362ns of delay is logical and 4.048ns of delay is route.

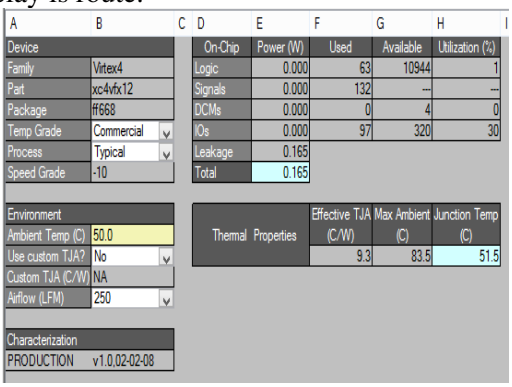


Figure 6.7. Power summary.

Figure 6. 7 shows the power consumption report of propsoed DST-R4BM. Here, the DST-R4BM consumed power as 0.165 watts.

7. CONCLUSION

It can be concluded that the design when scaled to higher bits only shows a marginal rise in delay due to its core strengths. Firstly, the parallelism involved in its partial product generation. Secondly, reduction of carry propagation delay in the novel adder it incorporates. Due to the use of QSD, the design is able to incorporate carry free arithmetic while eliminating radix conversion module speed overhead by integrating concept of adjusting bit logic in its architecture. The proposed design showed an increase in implementation area over some designs due to increased parallelism even in finer nuances of the architecture. The proposed design is targeted towards digital systems requiring high throughput and low latency at the cost of area overhead. For example, in a DSP system, operations such as Fast Fourier Transform, Convolution, Filtering and Discrete Wavelet transform etc. Multipliers play a key role in determining the speed of the system. Similarly, this architecture would be a good candidate to be implemented as a large part of systems like DCT, Central Processing Unit (CPU), MAC (Multiply and Accumulate) Unit, Image Processors where high-speed multiplications are critical to the performance of the system. It can also be observed that despite the objective of decreasing the delay, the proposed design performs better than most designs compared in terms of power for lower input bit sizes [16 and 32 bit]. Although it consumes more power than other designs higher input bit sizes [64 and 128 bit], it is justifiable when factored in with advantages gained in speed for higher input bits.

FUTURE SCOPE

we will extend an optimization for binary radix-32 (modified) Booth recoded multipliers to reduce the maximum height of the partial product columns to [n/4] for n = N-bit unsigned operands. This is in contrast to the conventional maximum height of [(n + 1)/4]. Therefore, a reduction of one unit in the maximum height is achieved. This reduction may add flexibility during the design of the pipelined multiplier to meet the design goals, it may allow further optimizations of the partial product array reduction stage in terms of area/delay/power and/or may allow additional addends to be included in the partial product array without increasing the delay. The method can be extended to Booth recoded radix-8 multipliers, signed multipliers, combined signed/unsigned multipliers, and other values of n.

REFERENCES

[1] I. Blake, G. Seroussi, andN.P.Smart, Elliptic Curves in Cryptography,ser. London Mathematical Society Lecture Note Series.. Cambridge,U.K.: Cambridge Univ. Press, 1999.

[2] N. R. Murthy and M. N. S. Swamy, “Cryptographic applications of brahmaqupta-bha skara equation,” IEEE Trans. Circuits Syst. I, Reg.Papers, vol. 53, no. 7, pp. 1565–1571, 2006.

[3] L. Song and K. K. Parhi, “Low-energy digit-serial/parallel finite field multipliers,” J. VLSI Digit. Process., vol. 19, pp. 149–C166, 1998.

[4] P. K. Meher, “On efficient implementation of accumulation in finite field over

GF(2^m) and its applications," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 17, no. 4, pp. 541–550, 2009.

[5] L. Song, K. K. Parhi, I. Kuroda, and T.Nishitani, "Hardware/software codesign of finite field datapath for low-energy Reed-Solomn codecs,"IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 8, no. 2, pp.160–172, Apr. 2000.

[6] G. Drolet, "A new representation of elements of finite fields GF(2^m) yielding small complexity arithmetic circuits," IEEE Trans. Comput.,vol. 47, no. 9, pp. 938–946, 1998.

[7] C.-Y. Lee, J.-S. Horng, I.-C. Jou, and E.-H. Lu, "Low-complexity bit-parallel systolic montgomery multipliers for special classes of GF(2^m)," IEEE Trans. Comput., vol. 54, no. 9, pp. 1061–1070, Sep. 2005.

[8] P. K. Meher, "Systolic and super-systolic multipliers for finite field GF(2^m) based on irreducible trinomials," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 55, no. 4, pp. 1031–1040, May 2008.

[9] J. Xie, J. He, and P. K. Meher, "Low latency systolic montgomery multiplier for finite field GF(2^m) based on pentanomials," IEEE Trans.Very Large Scale Integr. (VLSI) Syst., vol. 21, no. 2, pp. 385–389, Feb.2013.

[10] H.Wu, M. A. Hasan, I. F. Blake, and S. Gao, "Finite field multiplier using redundant representation," IEEE Trans. Comput., vol. 51, no. 11, pp. 1306–1316, Nov. 2002.

[11] A. H. Namin, H. Wu, and M. Ahmadi, "Comb architectures for finite field multiplication in F_{2m} ," IEEE Trans. Comput., vol. 56, no. 7, pp. 909–916, Jul. 2007.

[12] A. H. Namin, H. Wu, and M. Ahmadi, "A new finite field multiplier using redundat representation," IEEE Trans. Comput., vol. 57, no. 5, pp. 716–720, May 2008.

[13] A. H. Namin, H.Wu, and M. Ahmadi, "A high-speed word level finite field multiplier in F_{2m} using redundant representation," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 17, no. 10, pp. 1546–1550,Oct. 2009.